Resource management for multimedia applications, distributed in open and heterogeneous home networks

Maxime Louvel, Alain Plantec, Jean-Philippe Babau

Resource management for multimedia applications, distributed in open and heterogeneous home networks☆

Maxime Louvel[a,b,*], Alain Plantec[b], Jean-Philippe Babau[b]

[a] *France Telecom R&D, 28 chemin du vieux chêne 38240 Meylan France*
[b] *UBO Université Eurepéenne de Bretagne, UFR Sciences et Techniques Département Informatique, 20 avenue Le Georgie 29200 Brest France*

Abstract

The home network is an open, heterogeneous and distributed environment where ensuring multimedia applications' quality of service is a main concern. Mechanisms to reserve resources (CPU, memory, network) and architectures using them already exist. However, they require to modify the devices or the applications and they do not take into account the heterogeneity of the home network.

This paper presents a non-intrusive and adaptable resource management framework. This framework is developed upon an architecture, customized for the actual devices. This architecture uses global components that delegate to local components the management of local resources. These components rely on the resource reservation mechanisms provided by the Linux operating system in order to guarantee the resources to the applications.

The framework has been implemented on real devices (PCs, laptops and embedded multimedia devices), bridged with wireless and Ethernet networks. The evaluations of the framework show that reservations are guaranteed even if noise is generated on the resources, which also guaranties the expected quality of service.

Keywords: Home network, quality of service, resources management, multimedia, heterogeneity

1. Introduction

In the recent years, the home network has evolved from a few PCs connected to the Internet to an open environment, interconnecting devices that access distributed multimedia contents. Historically, telecoms service providers were the only one in the user home network. They are now sharing this market with devices manufacturers and applications providers. These three players provide the home with applications and devices that are automatically interconnected, thanks to standards such as UPnP (Universal Plug and Play) [3].

Thanks to these standards and to the improved embedded devices, multimedia application are now spreading through all the home devices. For instance, a user may want to render a movie, stored in a PC somewhere in its network, on its mobile phone using a Wifi link, or on the bedroom's Set-Top-Box using an Ethernet link.

Multimedia applications are considered as soft real time since violating a timing constraint does not lead to a system failure. However it degrades the Quality of Serivce (QoS) provided to the user. In the competitive home market its necessary to provide the expected QoS to the user. To provide such a QoS, a multimedia application requires a set of Quantities of Resource (QoR).

To guarantee these QoR, it is possible to reserve them to the applications [4, 5, 6]. During the last years, an effort has been done to build distributed resource management architecture [7, 8]. These architectures help the designer to make applications sensitive to resources availability. However these works focus on system or application design and they require to use specific patterns. Moreover these architecture assume specific mechanisms

☆This work is an extension of the work published in [1] and [2]. This paper presents a more global and integrated contribution, based on the work published in the two conferences. Moreover new evaluations are detailed.

*corresponding author

Email addresses: m.louvel@gmail.com (Maxime Louvel), firstname.lastname@univ-brest.fr (Jean-Philippe Babau)

such as Earliest Deadline First (EDF) scheduler to be available on the devices. Devices of the home network use General Purpose Operating Systems (GPOS) such as Linux. Even if research works have helped to integrate real-time scheduler inside Linux [9, 5, 10], a modification of the operating system is still required. In the home network neither the applications nor the devices may be modified easily. Finally these architecture are built using a specific middleware when none has emerged in the home network. Hence these solutions can not be used in the home network.

Besides the home network is an heterogeneous environment, in terms of devices, applications and communication technologies. This aspect has not been considered in the above solutions, limiting their ability to be used in this environment. Indeed, to be successfully used in the home network, a resource management solution needs to take into account this heterogeneity and to adapt itself to the devices, the communication technologies and the reservation mechanisms that are available.

1.1. Contributions of the paper

The first contribution of this paper is ARMOR (A Resource Management framewORk), to address these open issues. When a user asks to start a multimedia application, ARMOR checks that the required quantities of resource are available on all the resources used. In that case, it reserves the required quantities on all the resources using the operating system's mechanisms. Hence the application has the required QoR and provides the expected user-level QoS. ARMOR is based on a distributed resource management architecture to support streaming applications. This architecture relies on resource management components that are configured according to the operating system of the devices and the links used. Hence the components handle the home network heterogeneity. These components do the admission control tests and make the QoR reservations using standard Linux mechanisms and standard network protocols. Thus ARMOR supports home devices without modifying them. Finally, to support legacy application, ARMOR has a non intrusive approach to estimate and reserve the quantities of resources required by multimedia applications.

The second contribution of this paper is the implementation of ARMOR on real Linux based devices. To show its ability to be used in home networks, ARMOR has been tested on beagleboards[1] which are embedded multimedia devices, emulating typical home network devices such as mobile phones or Set-Top-Boxes.

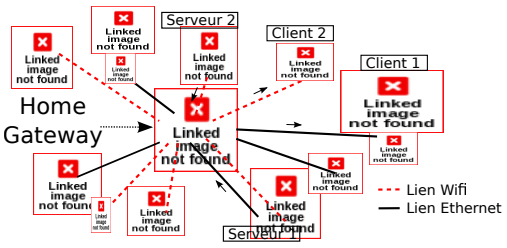


Figure 1: a home network example, with two multimedia applications

1.2. Paper structure

This paper is organized as follows. After introducing the context of this study, i.e. the home network, the section 2 details the reservation mechanisms available on the home device. After presenting related work on resource management, in section 3, section 4 presents our resource management framework called ARMOR. This section details the internals of ARMOR and how it integrates existing resource reservation mechanisms. Section 5 qualitatively evaluates ARMOR and shows it ability to guarantee the resources required by multimedia applications. Finally section 6 concludes this paper and gives future works.

2. Home network and resource reservation

Today's home networks are based on a star topology. The home gateway acts as a router (IP level), an Ethernet switch and a Wifi access point. The devices of this network are heterogeneous, and belong either to the user (mobile phone, PDA, PC) or to the telecoms service provider (gateway, Set-Top-Boxes).

The Figure 1 gives an example of a home network with two multimedia applications: from *server 1* to *client 1* and from *server 2* to *client 2*. A multimedia application consists in sending a given encoded stream, from the device executing the server to the device executing the client, via a given network link, with two given software.

[1] `http://beagleboard.org/`

Distributed multimedia applications use computing (CPU), storage (RAM) and communication resources (network interfaces and links). Resources belonging to a device (CPU, RAM and network interfaces) are managed by a GPOS. Among the existing GPOS, Linux is open and nowadays widely used; therefore this study focuses on Linux. The communication resources (the links and the home gateway) are shared among all the home devices. They are accessed through standard network protocols.

Before explaining how the resources are accessed and how they may be reserved, the next section gives definitions on quantity of resource reservation.

2.1. Quantity of Resource reservation

In order to reserve a quantity of resource to an application, it necessary to [11, 12, 13]:

- estimate the QoR required by the application and the QoR available on the resource
- do an admission control test to accept or reject the request
- guarantee and limit the reservation
- account for reservations' utilization.

If the four above requirements are respected, then two applications using the same resource will not interfere with each other, even if one is trying to use more than its reservation. This property is called temporal isolation [5].

Finally, there is three types of QoR reservation [5]:

- hard: an application can not use more than its reservation even if the resource is not used;
- firm: an application may use more than its reservation only if the resource is not used;
- soft: an application may use more than its reservation while this does not affect the other reservations.

However, in most of the work using QoR reservation, only hard and soft reservations are used.

This section now details the reservation mechanisms, that are available in the home network and that may be used to reserve the required QoR, starting with the network resources.

2.2. Network

A multimedia application uses three network resources: the network interfaces of the server and the client devices and the link between them. The link include the Ethernet cables and the home gateway or the Wifi link and the home gateway. It is not possible to control the traffic arriving to a device. Thus the reservations have to be done on the link and the interface of the server device.

Links are accessed through layered network protocols, typically RTSP (Real Time Streaming Protocol) over RTP (Real-time Transport Protocol) or UDP for streaming applications. RTP and UDP are used over the IP protocol, used over the protocol matching the underlying technology (Ethernet or Wifi) at the MAC layer. The home is a Local Area Network (LAN). In a LAN, the network bandwidth is managed at the MAC layer. Within this layer, network QoS standards define traffic classes and priority between traffic classes. For Ethernet, the standard is 802.1p, containing 7 traffic classes, and for Wifi the standard is 802.11e [14, 15], containing 4 traffic classes. Thanks to these standards the multimedia traffic accesses the link before the default traffic. However these standards do not differentiate the traffic of two multimedia applications. Hence two applications using the same link may interfere with each other. Dealing with this issue has to be done on the network interface of the server device. Finally, to use these standards, the server device and the home gateway have to support the same standard. Due to the heterogeneity and cost constraints this is not always the case in a home network. Once again this has to be dealt with at the network interfaces of the devices.

On the server device, operating system's mechanisms allow to differentiate the traffic of each multimedia application. In Linux, the Hierarchical Token Bucket (HTB) [16] elements of the traffic control tool (called *tc*)[2] offer scheduling and shaping of the exiting traffic. Scheduling assigns the output bandwidth to the different *tc* classes, while shaping limit the output of the *tc* classes. The shaping is done according to the token bucket model where the output traffic is defined by a mean rate r and a maximum burst b. Hence it is possible, on the server device, to reserve bandwidth to an application. This reservation is then guaranteed and the kernel also enforces that the application will not use more than its reservation.

[2] `http://lartc.org/`

2.3. CPU

There is two types of scheduling algorithms allowing for CPU reservation [17, 18]. The algorithms of the first type are based on a server and use a real time scheduler. One of the most used is the Constant Bandwidth Server (CBS) [19, 20] which is based on the well known Earliest Deadline First (EDF) scheduler. EDF executes the thread with the shortest deadline. A thread τ_i is assumed to have the parameters (C_i, T_i), where C_i is the Worst Case Execution Time (WCET) of the thread and T_i its activation period. With EDF, if the test (1) is true then the set of n threads is schedulable [21].

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \quad (1)$$

Legacy multimedia applications do not have explicit WCET and deadlines. Thus the CBS algorithm uses CBS servers with explicit parameters (C_s, T_s). A CBS server executes one or more threads. Hence the threads gain explicit WCET and deadlines. If the CBS server parameters match the thread parameters, then the thread deadlines are guaranteed [19]. The CBS scheduler allows soft or hard CPU reservation.

The other type of scheduling algorithms are called proportional share. They assign the CPU according to the weights of the threads. The default scheduler of the Linux kernel, the Completely Fair Scheduler (CFS) [22], is a proportional share scheduler. It ensures that after a period T_{sched}, each threads have its share of CPU F_i defined by (2), where w_i is the weight of the thread τ_i. By default T_{sched} is 20 milliseconds. If the number of threads grows too high, the kernel may increase T_{sched} in order to decrease the scheduler overhead. However in the context of this study, the number of threads executing stays low and T_{sched} is assumed to stay at 20 milliseconds. The CFS scheduler allows only soft CPU reservation.

$$F_i = \frac{w_i}{\sum_{j=1}^{n} w_j} \quad (2)$$

In addition, the scheduler CFS allows to manage the quantities of resource used by a group of threads, with the control groups system also called cgroups [23]. The share F_i defined in (2) is then used to assign the CPU between cgroups instead of between all the threads.

2.4. Memory

Multimedia and other applications require memory to work properly. When there is not enough RAM memory to serve all the threads currently executing, the operating system moves pages to the SWAP memory. The SWAP being slower than the RAM, it is necessary to prevent a multimedia application from using it. Otherwise deadline will be missed and the QoS will be degraded.

Similarly to the CPU, Linux offers a per cgroup memory management [24]. It allows to define a limit (soft or hard) on the RAM used by a cgroup. In the case of a hard limit, when a thread tries to use the RAM above its cgroup's limit, the memory management system moves pages belonging to the same cgroup to the SWAP. When a soft limit is used, a cgroup may use more of RAM than its limit while this memory is not claimed by another cgroup. Moreover, the Linux kernel move the memory to the SWAP on a per cgroup basis. If a thread of a cgroup uses a lot of memory, then pages belonging to a thread of the same cgroup are move to the SWAP. Thus the other cgroups are not move to SWAP.

This section has presented the reservation mechanism for the resources used by multimedia application. The next section presents related works, on Quantity of Resource management solutions integrating the above mechanisms.

3. Related works

This section details related works in the area of quantity of resource management. It first details works focusing on the architecture. Then it shows the solution that are integrated within the operating system, solutions focused on network resources and solutions integrated within the middleware. Finally it discusses the related works regarding the home network constraints.

3.1. Architecture

The research works focusing on the architecture used a centralized architecture with one global manager in the network and several local manager, installed on the devices. For instance the Quality-based Adaptive Resource Management Architecture (QARMA) [8], the Resource Allocation and Control Engine (RACE) [7] and the FRSH/-FORB [25] use this architecture and implement it

within a CORBA middleware. The same architecture is found at a local level only in Qinna [6] where a central manager handles all the contracts made to access the device's resources. The contracts for each resource are handled by a manager dedicated to the resource. Finally in the home network the UPnP consortium has defined the UPnP QoS specification [26]. This specification considers only the network resources but uses one global manager (called `QoSManager`) and several local managers (called `QoSDevice`) to manage the devices' network interfaces.

This architecture is suitable for a home network which is a closed environment with a relatively small number of devices (compare wide area networks). Moreover a home network belongs to a single user. Hence it is possible to control the devices and the links.

3.2. Operating system based solutions

Several approaches has been chosen integrate resource management in the operating system:

- a dedicated operating system such as Qinna [6];
- a strong modification of the Linux kernel to integrate a resource management architecture inside the kernel as in the resource kernel [5] or in Redline [27];
- a light modification of the Linux kernel, with a patch and a new kernel module as in AQuoSA [9].

A distributed extension to the resource kernel (distRK) [28] adds an admission control test for the end to end deadlines of the tasks, considering the network QoS characteristics (delay, jitter, loss rate). However there is no reservation made on the network to enforce the admission control test.

3.3. Network based solutions

In the home network community, research works have focused on providing enough of network bandwidth to multimedia application and guaranteeing the network QoS characteristics. These works do not focus on the whole problem of reserving the required quantities of resources as in this paper. In [29] the authors proposed a mapping between different network layers for stream coming from outside of the home network. In [30], the authors proposed a model based estimation of the available network bandwidth. In [31], the authors propose a joint admission and rate control. Their goal is to share the wireless bandwidth where we seek to enforce the bandwidth required by the multimedia applications. Finally, in [32], the authors propose the QoSiLAN system. QoSiLAN provides automatic discovery of the network and an automatic estimation of the required network bandwidth. The reservations are made by limiting the sending capacity of the devices. As far as the knowledge of the authors goes, their is no implementation of QoSiLAN.

3.4. Middleware based solutions

Several resource management solutions have been integrated into middlewares to support distributed applications. Traditional middleware are not efficient for real-time application. The related works detailed here use middleware such as TAO [33], CIAO [34] or QuO [35], that have brought real-time and adaptability features into CORBA. Relying on these middlewares, it is possible to make resource reservations for CORBA applications [36, 7]. The reservations are made thanks to real-time operating systems and specific network QoS standards.

Another promising approach is the FRSH/-FORB [25] project. It is implemented with a CORBA middleware but as far as our understanding goes it could be implemented within another middleware. FRSH/FORB focuses on the contract made to ensure the quantities of resources to the application. The contract can be done at several layers and for several resources. A contract is seen as a generic entity that is customized to fit a specific resource or different layer in their architecture. However FRSH/FORB does not focus on the heterogeneity of the available resource reservations mechanisms. For instance FRSH/FORB may support different operating systems but expects all of them to provide an EDF scheduler. For the network resources, it relies on the 802.11e network QoS standard.

3.5. Home network constraints

The table 1 summarizes the related works. The first column reference the solution, the second give the mechanisms used to guarantee the reservation, the third shows how good is the solution to use legacy applications. *** means the solution support legacy applications with known required QoR, ** means the applications have to modified and * means the applications have to be redesigned. The fourth column shows how close is a solution to be installed on existing devices. *** means the solutions

Solution	Guarantee	Leg.	Intr.	Res.
AQuoSA [9]	patch	***	**	C
Qinna [6]	OS	*	*	C,M
Redline [27]	OS	*	*	C,M,D
LinuxRK [5]	OS	***	**	C,M, D,N
distRK [28]	OS + AC	**	**	C
E-to-E QoS [36]	TimeSys DiffServ shaping	*	*	C,N
RACE [7]	TAO Diffserv RT-Linux	*	*	C,M,N
QARMA [8]	QuO, RTOS	*	*	C,M,N
FRSH/ FORB [25]	AQuoSA, 802.11e	***	***	C,D,N
UPnP QoS	AC	***	***	N
QoSiLAN [32]	bandwidth limit	***	***	N

Table 1: related works comparison (Leg.: Legacy application support: *** → possible, * → poor; Intr.: Intrusivity level: *** → low, * → high; Res. supported resource **C**PU, **M**emory, **D**isk, **N**etwork; AC: Admission Control)

can be use on a device if it provides the reservation mechanisms expected, ** means the Linux kernel needs to be slightly modified and * means it has to be strongly modified. Finally the last column gives the resources supported by the solution.

Only the three last lines have a double *** which means they are closer to be suitable for the home network. However FRSH/FORB makes strong assumption on the underlying operating system such as an EDF scheduler. Hence it can not be used easily in a home network. The two last solutions deal only with network resources. Moreover there is no implementation on actual devices of these solutions.

In addition, the solutions presented in this section do not consider the heterogeneity of the home network. They all assume specific reservation mechanisms to be there when they should adapt to the available mechanisms.

4. A Resource Management framewORk

This section details ARMOR, A Resource Management framewORk targeting quantity of resource management for multimedia applications, distributed in home networks. This section firstly describes how the quantities of resource are estimated for an application. Then it details how these quantities are expressed and what values are used for an application. Afterward, this section details the model of ARMOR and how it is integrated into a home network, thanks to its component based architecture. Then, this section describes the two stages used by ARMOR: an initialization stage, that is done only once, and a stage executed whenever a user wants to start a multimedia application. Finally, this section explains how the reservation are deleted when a multimedia application terminates.

4.1. Quantity of resource estimation

The quantity of CPU and RAM resources required on a device, for the client of the streaming (resp. the server) are function of:

- the content of the stream, its codec (e.g. H264) and its encoding parameters (e.g. frame size)
- the software used
- the device itself and the other resources used on the device (e.g. utilization of a Digital Signal Processor (DSP)).

To cope with this heterogeneity ARMOR uses a database filled with previous measurements. This database associates a required QoR to every configuration, on every resource. The size of the database may be decreased by aggregating the data [37] but this is not the scope of this paper. If an unknown configuration is requested, it is possible to execute the application with the biggest possible reservations on every resource and to monitor their usage for a future execution [38]. In this paper, we consider that the configuration is known when the user ask to start a multimedia application. Once ARMOR has read the necessary information into the database, it fills a manifest describing required quantities of CPU and RAM resources for the user request.

The required quantity of network resource is the same for all the network resources used. This quantity is directly related to the stream that is sent from the server to the client. To estimate the required network QoR, ARMOR analyzes the stream on-line, when the user requests to start the application, and computes the required network QoR [1].

4.2. Quantity of Resource expression

The QoR expression is function of the resource type. For the network resources the QoR is expressed, for an application i as a bandwidth X_i in

kilobits/s. Multimedia applications are also sensitive to the network QoS of the link. These requirements are expressed as constraints on the network QoS characteristics: the delay, the jitter and the loss rate of the link. For the CPU resources, the QoR is expressed with (C_{ik}, T_i) where C_{ik} is the quantity of CPU required by the application i on the device k, in % and T_i is the period of the application. Finally for the RAM resources, the QoR is expressed with a quantity Y_{ik} in Megabytes for the application i on the device k.

The quantities of resource used by an application fluctuates over the time. For the network resources, the streams used are encoded with a Variable Bit Rate (VBR). For the CPU resources, the quantity of CPU required to decode each frame, on the client, is also fluctuating. The quantity of RAM is more stable since almost all the memory allocations are done when the application is started. In order to simplify the admission control tests and the reservations, ARMOR uses a few values to express the required QoR on each resource, during the whole application's execution.

Network resources. Links, especially Wifi ones, are shared by all the devices in the home network. Over-reserving network resources, to ensure the quality of service, resources may be wasted and the number of applications that can run at the same time may decrease. However it is mandatory to ensure the QoS of multimedia applications. To deal with this trade-off, ARMOR makes two reservations for each application. One reservation, of $statMaxQoR_i$, is dedicated to the application. The other reservation, of $maxQoR_i - statMaxQoR_i$, is shared by all the applications using the same network resource. The required QoR for the network resources is expressed by:

- $maxQoR_i$: the maximum quantity of network required (in $kbits/s$).
- $statMaxQoR_i$: the maximum quantity of network required without the pics of utilization (in $kbits/s$). $statMaxQoR_i$ is the value where 10% of the required quantities are above and 90% are below. To compute $statMaxQoR_i$, the Bienaymé–Chebyshev [39] inequality is used. This inequality says that 89% of the values are under $arithmetic\ mean + 3\sigma$.[3]

For the network QoS of the link, three values are used to express an higher bound on every network QoS characteristic (delay, jitter, loss rate).

CPU and RAM resources. For the CPU resources, the reservation mechanisms do not allow to use a statMaxQoR value. Hence the required quantity of CPU is expressed, for an application i on a device k with $(maxQoR_{ik}, T_i)$ (in $(\%, milliseconds)$). The reservation period is set to $T_i = \frac{1}{frame\ rate_i}$, for the stream i, on the server and the client devices. Finally, for the RAM resources, the required QoR is $maxQoR_{ik}$ (in $MegaBytes$).

After this presentation of the QoR expressions, the internal of ARMOR is now detailed, starting with its model.

4.3. ARMOR's Model

ARMOR is made of resource management components called `Managers`. The Figure 2 shows the model of ARMOR using an UML class diagram. A `Manager` has a `type` and is either a `LocalManager` or a `GlobalManager`. The `types` currently supported are `CPU`, `RAM` and `network`. A `LocalManager` is in charge of managing one `LocalResource` of the same `type`. A `LocalResource` is a resource belonging to one device, such as the CPU, the RAM or a network interface. A `GlobalManager` coordinates the management of all the resources of the same type. It is directly in charge of managing the `GlobalResources` and it delegates the management of `LocalResources` to the corresponding `LocalManager`. A `GlobalManager` may manage several `GlobalResources` and delegate the management of several `LocalResources` while they are all of the same type. For instance, for the network type, the `GlobalManager` manages the links and delegates the management of the devices' network interfaces to `LocalManagers`. There is only one instance of a `GlobalManager` per resource type and there is only one instance of a `LocalManager` per resource on a device.

This section now details how this model is implemented and deployed in a home network.

4.4. ARMOR in a home network

All the `GlobalManagers` and all the `LocalManagers` on a device are grouped into composite components[4]. The composite for the

[3] this value makes no assumption on the data distribution. If it is a Gaussian, a lower value such as $arithmetic\ mean + 1.6\sigma$ may be used.

[4] a composite contains other components

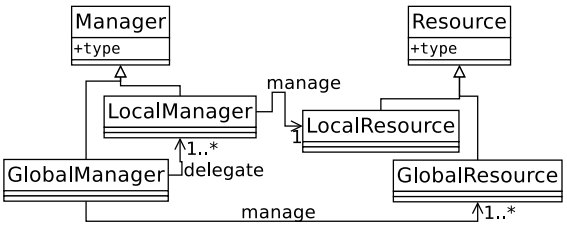


Figure 2: Elements of ARMOR

global managers is called `GRM` (Global Resources Managers) and the composites for the local managers are called `LRM` (Local Resources Managers). These composites are then installed on the home devices. The Figure 3 shows an example of the ARMOR's components for a home network with three devices. In this example, there is three instances of LRM: `LRM_1`, `LRM_2` and `LRM_3`, respectively installed on the three devices.

The `GlobalManager` in charge of the network, resp. the CPU and the RAM, resources is called `GNRM` (`Global Network Resources Manager`), resp. `GCRM` (`Global CPU Resources Manager`) and `GMRM` (`Global Memory Resources Manager`). On the local side, the `LocalManager` in charge of the network, resp. the CPU and the RAM resource of the device is called `LNRM` (`Local Network Resource Manager`), resp. `LCRM` (`Local CPU Resource Manager`) and `LMRM` (`Local Memory Resource Manager`).

The gray boxes in the Figure 3 represent composites. The details of these components is out of the scope of this paper, interested readers may look at [1, 2] to go into more details.

The Figure 3 also contains the component used to communicate with ARMOR: `console_1` and `cons_com_mgr` and the components to communicate between the `GlobalManagers` and the `LocalManagers`: `lrm_com_mgr` and `grm_com_mgr`. All the communication start with the `console_1` which forwards the requests to the `GRM`. The `cons_com_mgr` handles the request that is executed by the `GNRM`, the `GCRM` and the `GMRM`. The `lrm_com_mgr` and `grm_com_mgr` implement an RPC (Remote Procedure Call) like protocol to invoke methods of local managers, installed on other devices.

This component based architecture helps to handle the resource heterogeneity. All the components offers one QoR management interface containing four methods:

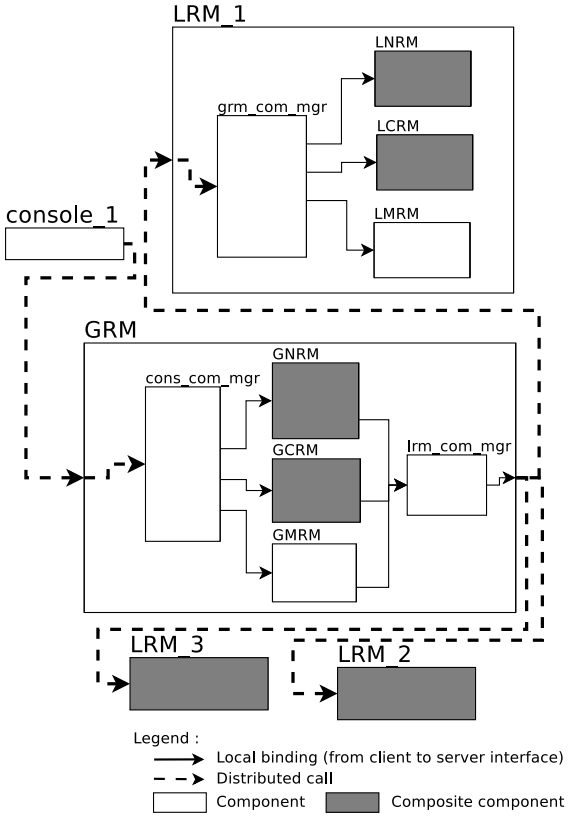


Figure 3: Example of an ARMOR's instance for a home network with 3 devices

	initialization	**admissionControl / reservation**
network	capacity network QoS	$maxQoR_i$ $statMaxQoR_i$
CPU	capacity, scheduler	$(maxQoR_{ik}, T_i)$
RAM	capacity	$maxQoR_{ik}$

Table 2: Parameters of the `LocalManager`' methods

- initialization()
- admissionControl()
- reservation()
- deletion(int id).

The first three methods, provided by the `GlobalManagers`' do not use any parameter in order to be as generic as possible. The deletion method only use the id of the reservation to be deleted. The QoR expressions, specific to each resource type, are only exposed by the `LocalManagers`. Since a `LocalManager`'s method is called only by the `GlobalManager` of the same type, they both know how to express the QoR for their type. The Table 2 gives the parameters of the `LocalManagers`' methods. The initialization method is called during the initialization stage that is detailed in the next section. The admissionControl and reservation methods are called when the user want to start an application, as detailed after the initialization stage. The deletion method is called with the same parameter on all the `LocalManagers`, thus it is not shown in Table 2. This method is detailed at the end of this section.

4.5. Initialization

ARMOR sees a device as a set of provided Quantities of Resources and an application as a set of required Quantities of Resources. Due to the heterogeneity of the home network, the operating system's resource management mechanisms differ from one device to the other. In order to hide this heterogeneity, the initialization stage, executed once, configures the `LocalManagers` components to use the operating system's mechanisms that are available on the devices. Once this stage is completed the user requests are handle independently of the underlying operating system. Hence, ARMOR may focus on the quantity of resource and not on the reservation mechanisms. This mapping is let to the component managing the resource. In addition to adapting the ARMOR components to the device, the initialization stage configures the resources' capacity and the reservation mechanisms offered by the operating system.

4.5.1. Capacity configuration

All the resources have a capacity that is the maximum quantity they may provide at all time. To reflect this, each `LocalManager` has a capacity attribute that is initialized as follows. For the network interfaces, it is estimated with an active measurement [1] because the actual capacity of a Wifi interface is lower than the standard. For the CPU, the capacity is set to $100\% * nb\ of\ CPUs$. Indeed ARMOR let the operating system handle the association of a thread to a CPU and consider a device to have one CPU with one capacity. Finally for the RAM, the capacity is set to the size of the RAM. For the links, managed by a `GlobalManager`, the capacity is set to the link standard (e.g. 45 Mbits/s for a 802.11g link).

4.5.2. Operating systems configuration

For the network interfaces, the `LocalManager` installs a network policy to differentiate the multimedia traffic and the default traffic. If the home gateway and all the network interfaces in the home support the same network QoS standard, then the `LocalManager` configures the policy to use a class with a higher priority for the multimedia traffic. The latter will not be bothered by default traffic. However, if a network interface or the home gateway does not support a network QoS standard, then ARMOR limits the default traffic on all the devices. Hence the data sent by any equipment will not bother the multimedia traffic. Finally the network managers (global and local) are configured to use either Ethernet of Wifi.

For the CPU, ARMOR supports two schedulers, one server based scheduler and one proportional share scheduler. The first scheduler is the Constant Bandwidth Server (CBS) that is integrated to the Linux kernel thanks to a patch [9]. The second scheduler is the Completely Fair Scheduler (CFS), that is the standard Linux scheduler. With CBS, the threads with no reservation are scheduled when no CBS server are ready to execute, thus no initialization is required. With CFS, the CPU is affected to the threads according to their weights. In order to control the quantity of CPU used by

the threads which do no belongs to a multimedia application, ARMOR creates two cgroups: the `user_group` and the `system_group`. All the user threads are then moved to the cgroup `user_group` and all the system threads are moved to the cgroup `system_group`. The weights of these cgroups are set to 1% of the CPU capacity to reserve them a small piece of the CPU. Since CFS only provides soft reservation, the user and system threads may use more of CPU while this does not affect the multimedia cgroups.

For the RAM, ARMOR also creates two cgroups for the user and system threads. The Linux kernel allows to limit the RAM utilization of a cgroup but not to reserve it. Hence ARMOR limits the utilization of the cgroups `user_group` and `system_group` in order to reserve the RAM for the multimedia application. The `system_group` is limited to 1% of the capacity, with a soft limit. Indeed, the system threads use most of the time a small part of the RAM; but they need to use more in a sporadic manner, for example for cronjobs of updates. The `user_group` is limited to the rest of the RAM capacity, with a hard limit.

Once the initialization stage has been completed for all the devices, a user may ask to start an application. The next section explains the steps done to answer this request.

4.6. Starting a multimedia application

When the user asks to start a multimedia application, via the `Global Resources Manager` (`GRM`), ARMOR firstly estimates the quantities of resource required on all the resources. Then the `GRM` asks to all the `GlobalManagers` to check if there is enough of resources available. If yes, the `GRM` triggers a reservation for all the resources used. This section details the admission control and reservation steps for each resource type, starting by the network. Then it details how an application is linked with the reservation made.

4.6.1. Network

The network resources are either global (links) or local to a device (network interface). The `GRM` lets the `GlobalManager` in charge of the network resources (the `Global Network Resources Manager` (`GNRM`)) handle all the network resources. The `GNRM` manages directly the global resources and delegates the management of the local resources to the `LocalManager` in charge of the network resource of the server device (the `Local Network Resource Manager` (`LNRM`)).

The admission control, for the link and the network interface of the server device, checks that the sum of all the dedicated reservations ($statMaxQoR_j$) done for the j applications currently using the resource and the shared reservation ($sharedQoR$) and the dedicated reservation of the new application ($statMaxQoR_i$) are lower than the resource's capacity, as show in (3).

$$capacity \geq \sum_{j=1}^{n} maxQoR_j + sharedQoR + maxQoR_i \quad (3)$$

The shared reservation is made for the highest pics of all the application using the resource, as show in (4) for n reservations, including the new application.

$$sharedQoR = max(\cup_{j=1}^{n}(maxQoR_j - statMaxQoR_j)) \quad (4)$$

On the reservation step, the `LNRM` adds the new reservation to the network policy installed on the server device. This network policy guarantees the required network quantity ($maxQoR_i$), to the application, on the network interface of the server device. It also limits the network usage of this application to $maxQoR_i$ in order to guarantee to reservation done on the link. The reservation and the limitation are implemented with the Hierarchical Token Bucket (HTB) elements of the traffic control (tc) tool available in Linux. Implementation details of this policy may be found in [1].

4.6.2. CPU

The CPU resources are local to a device. The `GRM` lets the `GlobalManager` in charge of the CPU resources (the `Global CPU Resources Manager` (`GCRM`)) handle all the CPU resources. The `GCRM` delegates the CPU management to the `LocalManagers` involved, i.e. for one multimedia application the `Local CPU Resource Manager` (`LCRM`) components of the server and the client devices.

The `LCRM` components support two schedulers: CBS and CFS. For CBS, the admission control test of EDF is used, including the new application as shown in (5) for j applications already using the resource. This test uses C_{ik} in milliseconds, while $maxQoR_{ik}$ is in %. To compute C_{ik} from

$maxQoR_{ik}$, the operation (6) is done.

$$\sum_{j=1}^{n} \frac{C_{jk}}{T_j} + \frac{C_{ik}}{T_i} < 1 \quad (5)$$

$$C_{ik} = \frac{maxQoR_{ik}}{capacity_k} * T_i \quad (6)$$

For CFS, the admission control test does not use the period of the application since the CFS scheduler only considers the scheduler period. Hence the admission control test checks that the CPU capacity is higher than the sum of the weights of the cgroup `user_group`, the cgroup `system_group`, the cgroups created for the j applications using this CPU and the required quantity of CPU for the new application i, as shown in (7).

$$capacity_k \geq w_{user\_group} + w_{system\_group} + \sum_{j=1}^{n} maxQor_{jk} + maxQor_{ik} \quad (7)$$

The CFS scheduler guaranties that each cgroup i has the CPU for a share F_i, proportional to its weight, as show in (2), every $T_{sched} = 20$ milliseconds. This equation is true no matter the weights' value. For example if 10 cgroups are created with a weight of 20 then, each one has share $F_i = \frac{20}{200} = 10\%$. If each cgroup requires 20% of the CPU then they do not have enough CPU. However, this situation is prevented by the admission control test and the sum of all the weights is never higher than the CPU's capacity. Hence all the cgroups have at least their required quantity of CPU.

On the reservation step, the `LCRM` component, of each device, make the reservation. For CBS, it creates a CBS server dedicated to the multimedia application with the parameters (C_{ik}, T_i). For CFS, it creates a cgroup dedicated to the multimedia application with a weight of $maxQoR_{ik}$.

Details on the internal of the CPU resources management components may be found in [2].

4.6.3. Memory

The RAM resources are local to a device. Similarly to the CPU, the `Global Memory Resources Manager` (`GMRM`) delegates the RAM management the `Local Memory Resource Manager` (`LMRM`) components of the server and the client devices.

As for the CPU with the scheduler CFS, the `LMRM` component uses the cgroups to manage the RAM accesses. During the initialization, two cgroups have been created: the `user_group` and the `system_group`. To make a reservation for a multimedia application, the `LMRM` component decrease the utilization limit of the cgroup `user_group`. The admission control test checks that the RAM capacity if higher than the sum of the reservations for the `system_group` and the quantities reserved for the j multimedia applications using this resource and the required RAM quantity for the application i, as shown in (8). The cgroup `user_group` has what is let free.

$$capacity_k \geq RAM_{system\_group} + \sum_{j=1}^{n} maxQoR_{jk} \quad (8)$$

The control cgroups for memory management do not allow to decrease the utilization limit of cgroup below its current utilization. If the admission control test is successful, the `LMRM` try to decrease the `user_group` limit. If the current memory usage allows this, the admission control is successful and the admission control is executed on the other resources. However if this action fails, the admission control test failed and the reservation is dropped.

On the reservation step, the `LMRM` creates a cgroup dedicated to the application and limits the memory usage of this cgroup to $maxQoR_{ik}$. The memory management system of Linux uses a variable called swappiness to determine when to start moving memory of a cgroup to the SWAP. A swappiness of 0 means the SWAP is used only when there is no more RAM available. For the multimedia applications, the limit is the maximum required memory. To prevent using the SWAP, the swappiness of the new cgroup is set to 0. Hence the multimedia application do not use the SWAP. The swappiness of the `user_group` and the `system_group` are let to the default value of 60.

4.6.4. Linking an application and its reservations

Once the application is started it needs to be bound to the reservations made. For the network resources, ARMOR uses the `IP_DEST` and `PORT_DEST` field of the IP header. For the CPU and RAM resources, ARMOR uses the PID[5] of the application on each device.

[5] Process ID

4.7. Deleting reservations

When an application ends, all its reservations have to be deleted. To identify all the reservations made for one application, ARMOR uses a global id. This id is generated by the `GRM` component, ensuring its uniqueness. When a manager (local or global) make a reservation it associate the reservation with the global id. Hence when an application ends, the `deletion` method is invoked on the `GRM` with the corresponding id. The `GRM` then asks all the managers to delete the associated reservation.

5. Evaluation

This section firstly evaluates ARMOR qualitatively. Then it details the experiments carried out to demonstrate the ability of ARMOR to guarantee the quantities of resource required by multimedia applications.

5.1. Qualitive evaluation

It is difficult to formally evaluate a framework. However, ARMOR has been implemented on real devices and this implements may be evaluated. ARMOR is implemented with the component based language Mind[6]. Mind is a C implementation of the Fractal component model [40] targeting embedded systems. Thanks to this language, the components of ARMOR still exists at run time. ARMOR is available on GPL[7] version 2 at `https://sites.google.com/site/mlouvel/software`.

The two main challenges arising from the home network are its heterogeneity and the intrusivity constraint on the resource management solutions. ARMOR handles the heterogeneity in its architecture with components dedicated to one type of resource (network, CPU, RAM memory). These components are configured during an initialization stage to hide the resource heterogeneity and the heterogeneity of the resource reservation mechanisms. On the intrusivity challenge, ARMOR relies only on standard Linux mechanism. Hence it may be install easily on Linux devices. Indeed, ARMOR has been tested on real Linux devices: PCs, laptops and beagleboards. The latter emulates traditional home network devices such as mobile phones of set-top-boxes. ARMOR is also not intrusive regarding the multimedia applications. Indeed it is only concerned by their resources requirements and the application are linked to the reservations without modifying them.

[6] `http://mind.ow2.org/`

[7] General Public License

	$\mathbf{L_i}$	$\mathbf{BB_1}$	
configuration	***c***	***c1***	***c2***
CPU	Intel(R) Core(TM) 2 Duo 1.06GHz	ARM Cortex-A8 700 Mhz	
scheduler	CFS	CFS	CBS
RAM (MB)	2000	256	
interface	Wifi G	Wifi G	
network QoS	802.11e	no	
distribution Linux	Ubuntu maverick	Ubuntu maverick	
noyau	standard 2.6.35-25	standard 2.6.35-25	AQuosA 2.6.32

Table 3: Features and configurations of devices

This evaluation shows that ARMOR may be used in a home network.

The rest of this now section evaluates the main goal of ARMOR: to guarantee the quantities of resources required by the application in order to guarantee their quality of service.

5.2. Quantities of resource guaranty

To evaluate the resource reservation made by ARMOR, three executions of the same application, with the same stream are done:

1. reference execution: the application is started and nothing else is running on the devices used;
2. noise, no reservation: the application is started and noise is generated on one of the resources used;
3. noise + ARMOR: a reservation is made by ARMOR, the application is started and noise is generated on one of the resources used.

The Table 3 gives the device used to carry out the evaluations detailed in this section. This table contains several laptops (L_i) running a completely standard ubuntu/Linux distribution. These laptops support the Wifi network QoS standard: 802.11e. The table also contains a beagleboard (BB_1) with two configurations. The configuration $c1$ is a standard ubuntu/Linux distribution, while $c2$ has a modified Linux with a CBS scheduler. The beagleboard doest not support any network QoS standard.

This section first summarizes the evaluations results when noise is generated on one resource, starting with the CPU. Then, this section offers a discussion.

5.2.1. CPU

To generate noise on the CPU resources, the tool `stress`[8] is used with `-c 10` to start 10 threads. Each thread uses as much of CPU as possible to compute random squared roots. The CPU used by the streaming client is measured with *top*. Top runs with the highest real-time priority to avoid being disturbed by the stress. The top delay is set to 400 ms in order to use a reasonably low quantity of CPU for the measure.

In the evaluations described here, the client of the streaming is started on the beagleboard BB_1, using the Wifi link. The stream used is "big buck bunny", that has been re-encoded from original file available at `http://www.bigbuckbunny.org/` with *x264*[9], a free *h264* encoder. Only the video stream is used. It is encoded with the baseline profile, a resolution of 352x288 (CIF format) and has a frame rate of 25 frames per second. The stream duration is 596 seconds. To decode the video, `mplayer` is used. The video decoding on the beagleboard is done entirely with the CPU.

Two evaluations are proposed for the CPU resources. The first one is focused on one multimedia application, for the three executions (reference, noise no reservation and noise + ARMOR). The second evaluation adds a local multimedia application on the streaming client, to be closer to the device's CPU capacity.

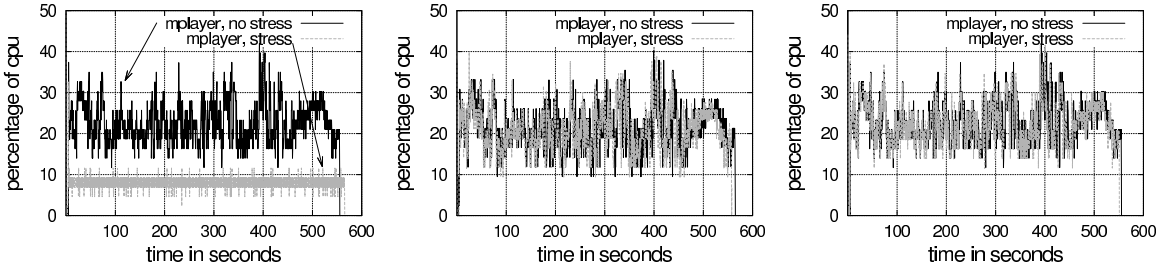
One multimedia application. The Figure 4 gives the quantity of CPU used by `mplayer` for the three executions. The Figure 4a compares the results of the executions 1 (reference: black curve) and 2 (noise, no reservation: gray curve). By default the Linux scheduler affects the CPU fairly between all the threads. Hence, in execution 2, `mplayer` gets a nearly constant quantity of CPU of about 10%, as do all the threads of the stress process. The Figure 4b compares the results of the executions 1 (black curve) and 3 (gray curve) for the configuration $c2$. In this configuration a CBS scheduler is available on the device. This scheduler affects the CPU according to the threads deadline, as it is traditionally done to make CPU reservations. On this figure, the two curves are almost merged, meaning that `mplayer` gets the same quantity of CPU for the two executions; it is not bothered by the stress. The Figure 4c compares the results of the executions 1 (black curve) and 3 (gray curve) for the configuration $c1$. In this configuration the default Linux scheduler, a proportional share scheduler is used. The reservation is made with the cgroups and their weights. As for configuration $c2$, the two curves are almost merged, meaning that `mplayer` gets the same quantity of CPU in the two executions.

These experiments shows that the same results are observed with the two schedulers. More details on these experiments and its impact on the application's QoS may be found in [2].

One streaming application and a local application. The reservation for the CFS scheduler are made with the weights of the cgroups. During the initialization stage, ARMOR creates two cgroups `user_group` and the `system_group` with a weight of 1% of the CPU capacity. For instance, the beagleboard BB_1 has a capacity of 100, thus each cgroup has a weight of 1. When a multimedia application is started a dedicated cgroup is created with a weight of $maxQoR_{ik}$ on the device k. The CFS scheduler then guarantees that after a period of time T_{sched}, the multimedia application had the CPU $maxQoR_{ik}$ times more than the `user_group`. For instance, for the stream used $maxQoR_{ik} = 45$. Hence ARMOR makes a reservation of 98% for the streaming client. To evaluate the CFS with a lower reservation, a local application is started on the beagleboard BB_1 together with the client of the streaming application. This local application is also `mplayer` and it decodes the same stream, stored locally. It requires the same quantity of CPU: 45%. Hence in this experiment, the sum of all the reservation is $45 + 45 + 1 + 1 = 92\%$ which is closer to the CPU capacity.

This evaluation makes four executions. Each time the streaming application and the local application are concurrently executed. First, the three executions detailed in the previous evaluation are done and the local application is executed with no reservation. For the forth execution, noise is generated and a reservation is made for the two applications (streaming and local). The Figure 5 gives the CPU used by the local application in the four executions. The CPU used by the streaming application

[8] `http://weather.ou.edu/~apw/projects/stress/`
[9] `http://www.videolan.org/developers/x264.html`



(a) without ARMOR (b) with ARMOR, configuration $c2$ (c) with ARMOR, configuration $c1$

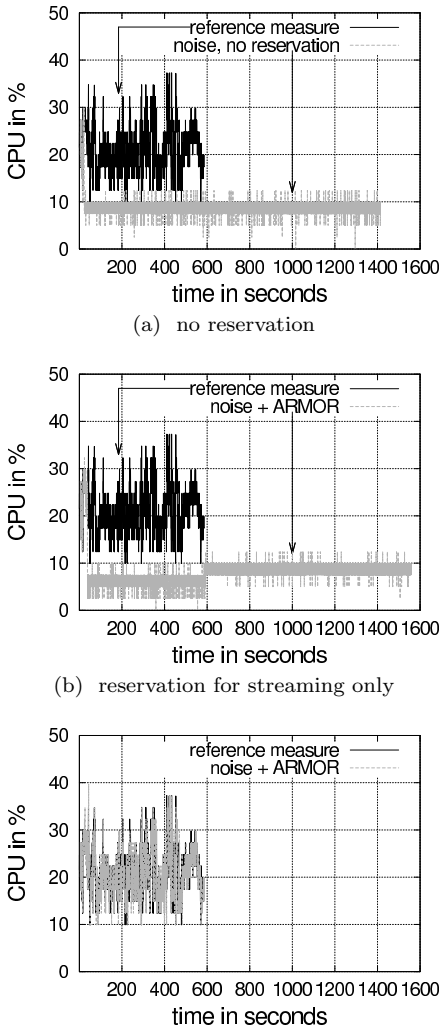
Figure 4: Quantity of CPU used by `mplayer`

is the same as previously detailed in Figure 4.

The Figure 5a compares the CPU used by the local application for the executions 1 (reference: black curve) and 2 (noise, no reservation: gray curve). As before, the CFS scheduler affects the CPU fairly between all the threads and the local `mplayer` has constantly less than 10% of the CPU. When `mplayer` decodes a local stream, if it does not have enough CPU it does not figure it out. Hence it jumps clumsily from one frame to the next and the video is jerky. This is shown on the figure, where the gray curve continues after 600 seconds, when the decoding should have stopped as in the reference execution. The Figure 5b compares the executions 1 (black curve) and 3 (noise + ARMOR for the streaming application only: gray curve). For the first 600 seconds, the two applications decodes the stream on the client. The streaming application has a reservation and is not bothered by the stress; the local application has a lower quantity of CPU. After 600 seconds, the streaming application ends. Afterward, the local application continue to execute for a while with around 10% of the CPU, due to the stress on the CPU. Finally the Figure 5c compares the execution 1 (black) and the forth execution (a reservation is made for both application). In this figure the two curves are merged showing that the local application is not bothered by the stress. As in the execution 3 the streaming application is not bothered either.

This evaluation shows that even if the sum of the reservation is close to the CPU capacity, the reservations made by ARMOR guaranty the required quantity of CPU to the multimedia applications. It has also evaluate that several reservations can successfully be done on the same resource at the same time.

The Table 4 evaluates the quality of service of the



(a) no reservation

(b) reservation for streaming only

(c) reservation for streaming and local applications

Figure 5: Quantity of CPU by the local application

execution	number of errors	ratio $\frac{\text{nb errors}}{\text{nb frames}}$
reference	66	0.005
noise, without ARMOR	6691	0.47
noise, with ARMOR, one reservation	67	0.005
noise, with ARMOR, two reservations	76	0.005

Table 4: Number of errors raised by mplayer for the streaming application

streaming application in the four executions. The first columns contains the execution, the second column is the number of errors raised by the client of the streaming application and the third column is the ratio $\frac{\text{nb errors}}{\text{nb frames}}$. When there is no reservation for the streaming application (line 2), the number of errors increases due to the generated noise. When a reservation is made (line 3 and 4) the number of errors is almost the same as in the reference execution (line 1). If these observations are linked to the quantity of resource observation made on the Figure 4, we can conclude that the reservations made by ARMOR guarantee the application's quality of service.

5.2.2. RAM

The evaluation for the RAM uses the same setup as the first evaluation of the CPU, described in section 5.2.1. This time the tool `stress` is used to generate noise on the RAM memory. To generate memory noise, stress has two parameters: the number of threads and the quantity of memory used by each thread. The noise also uses a lot of CPU. To avoid bias in the evaluation, a CPU reservation is always made by ARMOR. This section details two evaluations. For the first one, the streaming client is executed on the beagleboard BB_1 that has 256 megabytes of RAM and do not use the SWAP. For the second one, the streaming client is executed on the laptop L_1 that has 2048 megabytes of RAM and has a SWAP of 1000 megabytes.

Stress on the Beagleboard. The beagleboard has a low CPU capacity. In this evaluation, to avoid using too much of CPU, the streaming client is started with ssh and the window manager is not used.

On the beagleboard, when there is not enough RAM memory to serve all the processes, the Linux kernel has to kill one process to free memory.

In our experiments, when stress is called with 7 threads, each of them using 32M, the Linux kernel usually kills the stress. However there is no guarantee to this. When 10 stress process are started, each with one thread using 32M, the Linux kernel randomly kills some stress or `mplayer`. When ARMOR makes a reservation, the Linux kernel always kills a stress process or another process in the `user_group`. Indeed the RAM is managed on a per cgroup basis and all the stress are started in the `user_group`. Hence, with ARMOR, the multimedia application is not bothered by the stress.

Stress on the laptop. In this experiments, the window manager is used and the streaming client is it not started by ssh.

On the laptop, when there is not enough RAM memory to serve all the processes, the Linux kernel moves one process to the SWAP. This may have two consequences: slowing down the multimedia application, which degrades its QoS, and slowing down the whole system.

To test this, a stress is called with 8 threads, using 256 megabytes of memory, that is a total usage of 2048 megabytes or the RAM capacity. Hence to serve all the stress, all the other processes have to be moved to the SWAP. When this happens the device becomes unusable due to a huge response time. When ARMOR is used, a stress in the `user_group` still has a strong impact on the whole system. Indeed, this cgroup contains all the gnome processes. When the stress eats all the RAM, the other gnome processes are moved to the swap. Hence the device has a high response time which degrades the multimedia application even if the latter has enough of resources. However, if the stress is started within a dedicated cgroup with a limit of 1500 megabytes, then the device response time stays reasonable and the multimedia application's QoS is preserved.

Discussion. This evaluation has shown that when memory is heavily used, ARMOR is efficient only when the `user_group` does not contains important processes. This may be solved by using more cgroups than for the user processes. But this has not been implemented yet.

5.2.3. Network

ARMOR support both home network with and without network QoS support. An evaluation with the three executions, for Wifi networks, may be

found in [1]. This section focuses on the applications' quality of service evaluation, not detailed in previous paper.

To evaluate network reservation, noise is generated on the Wifi link used by the multimedia applications. To generate noise this noise, the software `iperf`[10] is used. `iperf` works with a client and a server. The `iperf` client sends data to the `iperf` server at a requested bandwidth, using the UDP protocol.

Two evaluations are now detailed, for a home network without and with network QoS support.

No network QoS. In this evaluation, the streaming server is started on a laptop and the client is started on the beagleboard BB_1. Noise is generated between four laptops, with four `iperf` servers and four `iperf` clients. The table 5b summarizes the network observations for the three execution. The first four lines gives statistics (max, mean, standard deviation and sum) on network bandwidth received on the streaming client. The four last lines gives statistics (max, mean and standard deviation) on the network delay observed. The third column of table is the reference execution (1), the forth if the noise no reservation execution (2) and the fifth is the noise + ARMOR execution (3).

The network bandwidth received on the streaming client is measured with `vnstat`[11] for the duration of the streaming application. The delay is measured with a `ping` from the streaming server to the streaming client. As shown in Table 5b, for the execution 2 the bandwidth measured on the streaming client is twice less than in the reference execution. However, when ARMOR is used (execution 3), the bandwidth measured is close to the reference measure. Similar results are observed on the delay. For execution 2, the delay is 500 time more important than in execution 1. In execution 3, the mean and standard deviation of the delay are twice higher than in execution 1 but stay low for a streaming application. Only the max value is too high. However a few values are not as representative as the mean and standard deviation.

The Table 6 evaluates the quality of service of the streaming application in the three executions. In the execution 2, the number of errors increased by a factor of 12.7 compare to the execution 1. When ARMOR makes a reservation for the application, the number of errors is only increased by a factor of 1.5 compare to the execution 1.

execution	**number of errors**	**ratio** $\frac{\text{nb errors}}{\text{nb frames}}$
reference	65	0.005
noise, no reservation	823	0.06
noise + ARMOR	92	0.006

Table 6: Number of errors raised by `mplayer`, no network QoS, big buck bunny

execution	**number of errors**	**ratio** $\frac{\text{nb errors}}{\text{nb frames}}$
reference	24	0.01
noise, no reservation	620	0.26
noise + ARMOR	24	0.01

Table 7: Number of errors raised by `mplayer`, network QoS, trailer Harry Potter 6

Network QoS support. For this evaluation the only the laptops are used. Hence ARMOR relies on the network QoS standard 802.11e, supported by all the devices, to make the reservations on the link. The stream used in this evaluation is the trailer Harry Potter 6[12].

6. Conclusion

This paper has presented ARMOR, a resource management framework to guarantee the Quantities of Resources (QoR) required by multimedia applications in home networks. The main challenges of this context are: support of legacy applications and existing devices, and the heterogeneity of the home network regarding resources and resource reservation mechanisms.

ARMOR guarantees the required QoR by reserving them in advance, when the application is started. ARMOR integrates existing resource management mechanisms, provided by the Linux operating system, in order to support existing devices. It is designed with a component based architecture, configured to adapt to the devices. The current version of ARMOR supports Ethernet and Wifi network, with and without network QoS support. It also support two kind of schedulers: the Constant

[10] `http://iperf.sourceforge.net/`

[11] `http://humdi.net/vnstat/`

[12] available on `http://trailers.apple.com/` The stream used is the HP6_Large.mov

	value
bandwith (kbits/s)	max
	mean
	std dev
	sum
delay (ms)	max
	mean
	std dev

(a)

reference	noise, no reservation	noise + ARMOR
1210	588	1120
257	116	236
204	136	206
91600	43812	92100
116	5320	632
3.82	1410	7.93
11.1	1710	30.0

(b) No network QoS, big buck bunny

reference	noise, no reservation	noise + ARMOR
5330	5170	4110
1180	425	608
881	728	833
66000	48000	68600
298	5530	510
4.53	880	8.81
20.2	1010	30.7

(c) Network QoS, trailer harry potter 6

Table 5: Network utilization statistics

Bandwidth Server and the Completely Fair Scheduler, which is the default Linux Scheduler. Regarding the applications, ARMOR has a non intrusive approach, considering only their required QoR and not their design.

ARMOR has been implemented on real devices, especially on beagleboards which are embedded devices emulating typical home network devices. To evaluate ARMOR, noise has been generated on the network links, the CPU and the memory of the devices. These evaluations have shown that ARMOR guarantees the required QoR which guarantees the user-level QoS, measured with the number of errors raised by the client of the streaming application.

ARMOR focuses on the QoR required by the applications, it does not consider the elements shared by multimedia applications and other applications of the devices. A typical example is the X windows server. Indeed if the X server does not have enough of CPU or RAM to execute, the user-level QoS is degraded even if the multimedia application has enough of CPU and RAM. This has been enlightened in the RAM evaluation. Future works will dig into reserving resources for these shared elements.

ARMOR reserves the QoR prior to the beginning of a multimedia application. A reservation is made only if all the admission control tests are successful. These tests compare the required QoR to the current available QoR on the resource, i.e. the resource capacity minus the reservation done. Afterward ARMOR assumes that the resource capacity does no change. This may be erroneous for Wifi links or battery powered devices. Hence future works will also focus on integrating adaptation to the fluctuation of the available QoR. This aspect has already been considered in other research works. However these works focused only on the adaptation part and not on guaranteeing the application's QoS by reserving the QoR. Future work on ARMOR will tackle this trade-off.

References

[1] M. Louvel, P. Bonhomme, J.-P. Babau, A. Plantec, A network resource management framework for multimedia applications distributed in heterogeneous home networks, in: Poceedings of the IEEE International Conference on Advanced Information Networking and Applications (AINA), 2011, pp. 724 –731.

[2] M. Louvel, J. Tous, A. Plantec, J.-P. Babau, Ensuring qos of multimedia applications in heterogeneous home networks: the cpu use case, in: IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC), 2011, to be published.
URL `https://sites.google.com/site/mlouvel/publications-2/cpu_euc_2011.pdf?attredirects=0&d=1`

[3] UPnP, www.upnp.org.

[4] K. Nahrstedt, J. Smith, The qos broker [distributed multimedia computing], Multimedia, IEEE 2 (1) (1995) 53 –67.

[5] R. Rajkumar, K. Juvva, A. Molano, S. Oikawa, Resource kernels: A resource-centric approach to real-time and multimedia systems, in: Conference on Multimedia Computing and Networking, 1998, pp. 150—164.

[6] J. Tournier, J. Babau, V. Olive, Qinna, a component-based qos architecture, in: Component-Based Software Engineering, 8th International Symposium, St. Louis, USA, Springer, 2005, pp. 107–122.

[7] N. Shankaran, N. Roy, D. C. Schmidt, X. D. Koutsoukos, Y. Chen, C. Lu, Design and performance evaluation of an adaptive resource management framework for distributed real-time and embedded systems, EURASIP J. Embedded Syst. 8 (3) (2008) 1–20.

[8] D. Fleeman, M. Gillen, A. Lenharth, M. Delaney, L. Welch, D. Juedes, C. Liu, Quality-based adaptive resource management architecture (QARMA): a CORBA resource management service, in: proceedings of the 18th International Parallel and Distributed Processing Symposium, 2004, p. 116.

[9] T. Cucinotta, L. Palopoli, L. Marzario, G. Lipari, AQuoSA—adaptive quality of service architecture, Software: Practice and Experience 39 (1) (2009) 1–31.

[10] L. Abeni, C. Scordino, G. Lipari, L. Palopoli, Serving non real-time tasks in a reservation environment, in: proceedings of the 9th Real-Time Linux Workshop, 2007.

[11] C. Mercer, S. Savage, H. Tokuda, Processor capacity reserves: operating system support for multimedia applications, in: Proceedings of the International Conference on Multimedia Computing and Systems, 1994, pp. 90 –99.

[12] L. Steffens, G. Fohler, G. Lipari, G. Buttazzo., Resource reservation in real-time operating systems - a joint industrial and academic position, in: International Workshop on Advanced Real-Time Operating System Services (ARTOSS), 2003, pp. 25–30.

[13] R. M. Laverty, Robust open component based software architecture for configurable devices project, Tech. rep., Information Technology for European Advancement (2003).

[14] D. Gao, J. Cai, K. Ngan, Admission control in IEEE 802.11e wireless LANs, IEEE network 19 (4) (2005) 6–13.

[15] C. Liu, C. Zhou, Providing quality of service in ieee 802.11 wlan, in: Proceedings of the Advanced Information Networking and Applications, 2006, pp. 817–824.

[16] M. Devera, Hierachical token bucket theory, `http://luxik.cdi.cz/devik/qos/htb/manual/theory.htm` (2002).

[17] L. Abeni, G. Lipari, G. Buttazzo, Constant bandwidth vs. proportional share resource allocation, in: proceedings of the IEEE International Conference on Multimedia Computing and Systems, Vol. 2, 1999, pp. 107 –111 vol.2.

[18] G. F. (TUKL), A. N. (TUKL), K.-E. Årzén (ULUND), C. L. (EPFL), M. M. (EPFL), V. N. (AKAtech), C. von Platen (Ericsson), G. B. (SSSA), E. B. (SSSA), C. S. (EVI), State of the art assessment, Tech. rep., ACTORS Adaptivity and Control of Resources in Embedded Systems (2008).

[19] L. Abeni, G. Buttazzo, Integrating multimedia applications in hard real-time systems, in: proceedings of the 19th IEEE Real-Time Systems Symposium, 1998, pp. 4–13.

[20] L. Abeni, G. Buttazzo, Resource reservation in dynamic real-time systems, Real-Time Systems 27 (2004) 123–167.

[21] C. L. Liu, J. W. Layland, Scheduling algorithms for multiprogramming in a Hard-Real-Time environment, J. ACM 20 (1) (1973) 46–61.

[22] C. Pabla, Completely fair scheduler, Linux Journal 2009 (184) (2009) 4.

[23] P. Menage, Adding generic process containers to the linux kernel, in: Proceedings of the Linux Symposium, Vol. 2, 2007, p. 45–57.

[24] J. Corbet, Controlling memory use in containers, `http://lwn.net/Articles/243795/` (2007).

[25] M. Sojka, P. Písa, D. Faggioli, T. Cucinotta, F. Checconi, Z. Hanzálek, G. Lipari, Modular software architecture for flexible reservation mechanisms on heterogeneous resources, Journal of Systems Architecture 57 (4) (2011) 366 – 382.

[26] UPnP^{TM}, Upnp-qos architecture v3.0, Tech. rep., UPnP^{TM} Forum (2008).

[27] T. Yang, T. Liu, E. D. Berger, S. F. Kaplan, J. E. B. Moss, Redline: first class support for interactivity in commodity operating systems, in: Proceedings of the 8th USENIX conference on Operating systems design and implementation, OSDI'08, USENIX Association, Berkeley, CA, USA, 2008, pp. 73–86.

[28] K. Lakshmanan, R. Rajkumar, Distributed resource kernels: OS support for End-To-End resource isolation, in: proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium, 2008, pp. 195–204.

[29] F. Cuomo, An Architectural Model to Provide QoS in a Home Network and its Evaluation in a Real Testbed, Journal of Networks 3 (6) (2008) 44–53.

[30] D. Ko, S. Han, H. Cha, R. Ha, A traffic control system for IEEE 802.11 networks based on available bandwidth estimation, Wireless Communications and Mobile Computing 8 (4) (2008) 407–419.

[31] R. Yu, Y. Zhang, C. Huang, R. Gao, Joint admission and rate control for multimedia sharing in wireless home networks, Comput. Commun. 33 (14) (2010) 1632–1644.

[32] C. Kohnen, C., V. Rakocevic, M. Rajarajan, R. Jager, Qosilan - a heterogeneous approach to quality of service in local area networks, in: proceedings of Advances in Multimedia, International Conference on, IEEE Computer Society, 2010, pp. 109–112.

[33] D. C. Schmidt, D. L. Levine, S. Mungee, The design of the tao real-time object request broker, Computer Communications 21 (1998) 294—324.

[34] N. Wang, C. Gill, Improving Real-Time system con.guration via a QoS-Aware CORBA component model, in: Proceedings of the 37th Annual Hawaii International Conference on System Sciences - Track 9 - Volume 9, IEEE Computer Society, 2004, p. 90273.2.
URL `http://portal.acm.org/citation.cfm?id=963298`

[35] R. Vanegas, J. A. Zinky, J. P. Loyall, D. Karr, R. E. Schantz, D. E. Bakken, Quo's runtime support for quality of service in distributed objects, in: Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, Middleware '98, Springer-Verlag, London, UK, 1998, pp. 207–222.

[36] P. Manghwani, J. Loyall, P. Sharma, M. Gillen, J. Ye, End-to-end quality of service management for distributed real-time embedded applications, in: Proceedings. 19th IEEE International Parallel and Distributed Processing Symposium, IEEE, 2005, p. 138a.

[37] M. Louvel, J. Pulou, A. Plantec, J.-P. Babau, Quantity of resource aggregation for heterogeneous resource reservation for multimedia applications, in: Work In Progess session of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2010, pp. 1–4.

[38] C. El Kaed, L. Petit, M. Louvel, A. Chazalet, Y. Denneulin, F.-G. Ottogalli, Insight: Interoperability and service management for the digital home, in: Proceedings of the 12th International Middleware Conference Industrial track, Middleware Industrial Track, ACM, New York, NY, USA, 2011, to be published.
URL `https://sites.google.com/site/mlouvel/publications-2/Middleware2011-Industrial.pdf?attredirects=0&d=1`

[39] S. Ghahramani, Fundamentals of probability(2nd Edition) , Prentice Hall, 2000.

[40] G. Blair, T. Coupaye, J. Stefani, Component-based architecture: the Fractal initiative, Annals of Telecommunications 64 (1) (2009) 1–4.